

EXAMPLE:

```
10 OPEN "O",1,"LIST
20 LINE INPUT "CUSTOMER INFORMATION?" ;C$
30 PRINT #1, C$
40 CLOSE 1
50 OPEN "I",1,"LIST"
60 LINE INPUT #1, C$
70 PRINT C$
80 CLOSE 1
RUN
CUSTOMER INFORMATION? LINDA JONES    234,4
    MEMPHIS
LINDA JONES    234,4    MEMPHIS
Ok
```

3

LIST Command

FORMAT:

LIST [[<line no. > [- [<line no. >]]] [, <filespec >]]

PURPOSE:

Allows a program to be listed to the screen or other devices.

REMARKS:

<line no. > is a valid line number from 0 to 65529.

<filespec > is a valid string expression returning a valid file specification.

RULES:

1. If the optional parameter < filespec > is omitted, the specified lines are listed to the screen.
2. Listings directed to the screen by omitting < filespec > can be stopped at any time by pressing CTRL-C.
3. If the line range is omitted, the entire program is listed.
4. When the dash (-) is used in a line range, you have three options:
 - a. If only the first number is given, that line and all higher numbered lines are listed.
 - b. If only the second number is given, all lines from the beginning of the program through the given line are listed.
 - c. If both numbers are specified, the inclusive range is listed.

EXAMPLE:

```
LIST , "LPT1: "
```

Lists program to the Line Printer.

```
LIST 10-20
```

Lists lines 10 through 20 to the screen.

```
LIST 10- , "SCRN: "
```

Lists lines 10 through last to the screen.

```
LIST -200
```

Lists first through line 200 to the screen.

```
LIST 1000-1045, "COM1:4800,0,5,2"
```

Lists lines 1000 through 1045 to serial port A, setting the baud rate, parity, data bits, and stop bits.

LLIST Command

FORMAT:

LLIST [{ < line number > [-[< line number >]] [+ < line number >]}]

PURPOSE:

Lists all or part of the program currently in memory on the line printer.

REMARKS:

LLIST assumes a 132-character-wide printer.

VBASICA always returns to command level after an LLIST is executed. The options for LLIST are the same as for the LIST command.

EXAMPLE:

See the examples for the LIST command. With the exception of the last one, which addresses a device, LLIST works in a similar way.

LOAD Command

FORMAT:

LOAD " < filespec > " [,R]

PURPOSE:

Loads a program from the specified device into memory, and optionally runs it.

3

REMARKS:

< filespec > is a valid string expression for the file specification. In VBASICA 2.0 and later, it can contain a path. Refer to Chapter 1.5 for more information on file specifications.

LOAD closes all open files and deletes all variables and program lines currently residing in memory before it loads the designated program. However, if the "R" option is used with LOAD, the program is RUN after it is LOADED, and all open data files are kept open. Thus, you can use LOAD with the "R" option to chain several programs (or segments of the same program). Information may be passed between the programs using their disk data files.

EXAMPLE:

This statement allows you to enter programs from the communications ports:

LOAD "COM1:4800,0,7,1",R

LOC Function

FORMAT:

LOC(< file number >)

PURPOSE:

With random disk files, LOC returns the actual record number within the file.

With sequential files, LOC returns the current byte position in the file, divided by 128.

REMARKS:

< file number > is the number under which the file was opened.

When a file is opened for APPEND or OUTPUT, LOC returns the size of the file in (bytes/128).

For a communications file, LOC(X) determines if any characters are in the input queue waiting to be read. If more than 255 characters are in the queue, LOC(X) returns 255. Because strings are limited to 255 characters, this practical limit alleviates the need to test for string size before reading data into it.

If fewer than 255 characters remain in the queue, the value returned by LOC(X) depends on whether the device was opened in ASCII or binary mode. In either mode, LOC returns the number of characters that can be read from the device. However, in ASCII mode, the low level routines stop queueing characters as soon as end-of-file is received. The end-of-file itself is not queued and cannot be read. An attempt to read the end-of-file results in an "Input past end" error.

EXAMPLE:

```
200 IF LOC(1)>50 THEN STOP
```

LOCATE Statement

FORMAT:

LOCATE [**<row>**] [, [**<col>**] [, [**<cursor>**] [, [**<start>**]
[, [**<stop>**]]]]

PURPOSE:

Moves the cursor to the specified position on the active screen. Optional parameters turn the cursor on and off and define the start and stop scan lines for the cursor.

REMARKS:

<row> is the screen line number, a numeric expression returning an unsigned integer in the range 1 to 25.

<col> is the screen column number, a numeric expression returning an unsigned integer in the range 1 to 40 or 1 to 80, depending on the screen width.

<cursor> is a Boolean value indicating whether the cursor is visible, with 0 for off, nonzero for on.

<start> is the cursor starting scan line, a numeric expression returning an unsigned integer from 0 to 31.

<stop> is the cursor stop scan line, a numeric expression returning an unsigned integer from 0 to 31.

The LOCATE statement moves the cursor to the specified position. Subsequent PRINT statements begin placing characters at this location. Optionally it can be used to turn the cursor on or off or to change the size of the cursor.

Any values entered outside these ranges result in the “Illegal function call” error. VBASICA retains previous values. You can omit any of the parameters. Omitted parameters assume the previous value.

If the start scan line parameter is given and the stop scan line parameter is omitted, stop assumes the start value. If both are omitted, the start and stop scan lines retain their previous values.

EXAMPLE:

Move to the home position in the upper left corner:

```
10 LOCATE 1,1
```

Make the cursor visible; the position remains unchanged:

```
20 LOCATE , , 1
```

The cursor position and visibility remain unchanged. Set the cursor to display at the bottom of the character starting and ending on scan line 15:

```
30 LOCATE , , , 15
```

Move to line 5, column 1, turn cursor on; cursor covers entire character cell starting at scan line 0 and ending on scan line 9:

```
40 LOCATE 5,1,1,0,9
```

NOTE: Usually, VBASICA does not print to line 25. To put things on line 25, turn off the soft key display using KEY OFF, then use LOCATE 25,1 : PRINT...

LOF Function

FORMAT:

LOF(< file number >)

PURPOSE:

Returns the number of bytes allocated to the file.

REMARKS:

< file number > is associated with a currently open file. For diskette files, LOF returns a multiple of 128. For example, if the actual file length is 257 bytes, the number 384 is returned.

For communications, LOF returns the amount of free space in the input buffer. That is, size-LOC(filnum), where size is the size of the communications buffer, defaults to 256 but can be changed with the /C: option at VBASICA initialization time.

EXAMPLE:

```
10 OPEN "DATA.FIL" AS #1
20 GET #1, LOF(1)/128
```

These statements get the last record of the file, assuming the record length is 128 bytes.

```
10 OPEN "FILE.BIG" AS #1
20 GET #1, LOF(1)/128
```

These statements get the last record of the file FILE.BIG, assuming that the file was created with a default record length of 128 bytes.

LOG Function

FORMAT:

LOG(X)

PURPOSE:

Returns the natural logarithm of X. X must be greater than zero.

EXAMPLE:

```
PRINT LOG(45/7)
1.860752
Ok
```

3

LPOS Function

FORMAT:

LPOS(X)

PURPOSE:

Returns the current position of the printhead within the line printer buffer. LPOS(X) does not necessarily give the physical position of the printhead.

REMARKS:

X is a dummy argument.

EXAMPLE:

```
100 IF LPOS(X)>60 THEN LPRINT CHR$(13)
```

LPRINT and LPRINT USING Statements

FORMAT:

LPRINT [<expr list>]

LPRINT USING <string expr>; <expr list>

PURPOSE:

Prints data at the line printer.

REMARKS:

Same as PRINT and PRINT USING, except output goes to the line printer.

LPRINT assumes a 132-character-wide printer.

LSET and RSET Statements

FORMAT:

LSET <str var> = <str expr>

RSET <str var> = <str expr>

PURPOSE:

Moves data from memory to a random file buffer, in preparation for a PUT statement.

REMARKS:

If <str expr> needs fewer bytes than were allocated to the field containing <str var>, LSET left-justifies the string in the field, and RSET right-justifies the string. Spaces are used to pad the extra positions. If the string is too long for the field, characters are dropped from the right.

Numeric values must be converted to strings before they are LSET or RSET. See the MKI\$, MKS\$, and MKD\$ functions.

EXAMPLE:

```
150 LSET A$=MKS$(AMT)
160 LSET D$=DESC($)
```

LSET or RSET can also be used with a nonfielded string variable to left-justify or right-justify a string in a given field. For example, the program lines:

```
110 A$=SPACE$(20)
120 RSET A$=N$
```

right-justify the string N\$ in a 20-character field. This is valuable when you are formatting printed output.

MERGE Command

FORMAT:

MERGE <filespec>

PURPOSE:

Merges the lines from an ASCII program file into the program currently in memory.

REMARKS:

<filespec> is a string expression that returns a valid file specification. In VBASICA 2.0 and later releases, it can contain a path. Refer to Chapter 1.5 for more information on file specifications.

EXAMPLE:

MERGE "COM2:4800,0,7,1"

MID\$ Function and Statement

FORMAT:

MID\$(<str expr1 > ,n[,m]) = <str expr2 >

PURPOSE:

Replaces a portion of one string with another string.

REMARKS:

n and m are integer expressions.

<str expr1 > and <str expr2 > are string expressions.

Beginning at position n, the characters in <str expr1 > are replaced by the characters in <str expr2 >. The m option can be used to specify the number of characters from <str expr2 > in the replacement. If m is omitted, all of <str expr2 > is used. However, the number of characters replaced can never exceed the original length of <str expr1 >.

EXAMPLE:

```
10 A$="DIDDY WAH WAH"  
20 MID$(A$,11)="DIDDY"  
30 PRINT A$  
RUN  
DIDDY WAH DIDDY
```

MID\$ can also return a substring of a given string.

MKDIR Command

FORMAT:

MKDIR <pathname>

PURPOSE:

Creates a new directory.

3

REMARKS:

<pathname> is a string expression specifying the name of the directory to be created. MKDIR works exactly like the DOS command MKDIR. The <pathname> must be a string of less than 63 characters.

EXAMPLE:

Assume the current directory is the root. This example creates a subdirectory named SALES in the current directory of the current drive:

```
MKDIR "SALES"
```

The following example creates a subdirectory named USERS in the current directory of drive B:

```
MKDIR "B:USERS"
```

Also see the CHDIR and RMDIR statements.

MKI\$, MKS\$, and MKD\$ Functions and Statements

FORMAT:

MKI\$(<int expr >)

MKS\$(<single-precision expr >)

MKD\$(<double-precision expr >)

PURPOSE:

Converts numeric values to string values. Any numeric value put into a random file buffer using an LSET or RSET statement must be converted to a string. MKI\$ converts:

- ▶ An integer to a 2-byte string
- ▶ A single-precision number to a 4-byte string
- ▶ A double-precision number to an 8-byte string

EXAMPLE:

```
90 AMT=(K+T)
100 FIELD #1,8 AS D$,20 AS N$
110 LSET D$ = MKS$(AMT)
120 LSET N$ = A$
130 PUT #1
```

NAME Command

FORMAT:

NAME "<old filename>" AS "<new filename>"

PURPOSE:

Changes the name of a disk file.

3

REMARKS:

<old filename> must exist and <new filename> must not exist; otherwise an error results. The renamed file occupies the same area of disk space as it did under the old name.

EXAMPLE:

In the following example, the file formerly named ACCTS is renamed LEDGER:

```
Ok
NAME "ACCTS" AS "LEDGER"
Ok
```

NEW Command

FORMAT:

NEW

PURPOSE:

Deletes the program in memory and clears all variables.

REMARKS:

Enter NEW at command level to clear memory before entering a new program. VBASICA always returns to command level after executing a NEW command.

3

OCT\$ Function

FORMAT:

OCT\$(X)

PURPOSE:

Returns a string that represents the octal value of the decimal argument. X is rounded to an integer before OCT\$(X) is evaluated.

Use the HEX\$ function for hexadecimal conversion.

EXAMPLE:

```
PRINT OCT$(24)
30
Ok
```

ON COM Statement

FORMAT:

ON COM(<n>) GOSUB <line number>

PURPOSE:

Sets up a line number for VBASICA to trap when information is coming into the communications buffer.

3

REMARKS:

<n> is the number of the communications port, where 1 is port A and 2 is port B.

<line number> is the starting line number of the routine to handle the information coming from the port. A line number of 0000 (zero) disables trapping of communication for the specified port.

You must execute a COM(<n>) ON statement to activate this statement for port <n>. After COM(<n>) ON, if you specify a nonzero line number in the ON COM(<n>) statement, every time VBASICA starts a new statement it checks to see if any characters have come through the specified port. If so, it performs a GOSUB to the specified line number.

If you execute COM(<n>) OFF, no trapping occurs for the port and the event is not remembered even if it does take place.

If you execute a COM(<n>) STOP statement, no trapping can occur for the port and the event is not remembered even if it does take place.

If you execute a COM(<n>) STOP statement, no trapping can occur for the port. But if VBASICA receives a character, it is remembered and an immediate trap occurs when you execute COM(<n>) ON.

When the trap occurs, VBASICA creates an automatic COM(< n >) STOP. Consequently, recursive traps can never occur. The RETURN from the trap routine automatically does a COM(< n >) ON unless an explicit COM(< n >) OFF was performed inside the trap routine.

Event trapping does not occur when VBASICA is not executing a program. When an error trap (resulting from an ON ERROR statement) occurs, all trapping is automatically disabled (including ERROR, COM, and KEY).

Before returning to the main program, the communications trap routine typically reads an entire message from the communications port. At high baud rates, the overhead of trapping and reading for each character can overflow the communications buffer. Therefore, avoid using the communications trap for single-character messages.

See the RETURN statement for more information.

EXAMPLE:

```
100 PORT. A = 1
110 PORT. B = 2
120 ON COM(PORT. A) GOSUB 500
    .
    .
    .
500 '*****      ROUTINE TO HANDLE PORT A CHRS
    .
    .
    .
550 RETURN
```

ON ERROR GOTO Statement

FORMAT:

ON ERROR GOTO < line >

PURPOSE:

Enables error trapping and specifies the first line of the error-handling subroutine.

3

REMARKS:

< line > is the number of the first line of an error-handling subroutine.

After error trapping is enabled, all errors detected—including direct mode errors (for example, syntax errors)—cause VBASICA to jump to the specified error-handling subroutine. If < line > does not exist, an “Undefined line” error results.

To disable error trapping, execute an ON ERROR GOTO statement. Subsequent errors print an error message and halt execution. An ON ERROR GOTO in an error-trapping subroutine tells VBASICA to stop and print the error message for the error that caused the trap. All error-trapping subroutines should execute an ON ERROR GOTO if they encounter an error for which no recovery action exists.

If an error occurs while an error-handling subroutine is executing, the VBASICA error message is printed and execution stops. Error trapping does not occur within the error-handling subroutine.

EXAMPLE:

```
ERROR
10 ON ERROR GOTO 1000
```

ON...GOSUB and ON...GOTO Statements

FORMAT:

ON <expr> **GOTO** <list>

ON <expr> **GOSUB** <list>

PURPOSE:

Branches to one of several specified line numbers, depending on the value returned when <expr> is evaluated.

REMARKS:

<expr> is a numeric expression.

<list> is a list of line numbers.

The value of <expr> determines which line number in the list is used for branching. If the value is three, the third line number in the list is the destination of the branch. If the value is noninteger, the fractional portion is rounded.

In the ON...GOSUB statement, each line number in the list must be the first line number of a subroutine. If the value of <expr> is zero, or greater than the number of items in the list (but less than or equal to 255), VBASICA continues with the next executable statement. If the value of <expr> is negative or greater than 255, an "Illegal function call" error occurs.

EXAMPLE:

```
100 ON L-1 GOTO 150,300,320,390
```

ON KEY(n) Statement

FORMAT:

ON KEY(<n>) GOSUB <line number>

PURPOSE:

Sets up a line number for VBASICA to trap when you press the specified function key or cursor control key.

3

REMARKS:

<n> is a numeric expression returning a value between 1 and 20 and indicates the key to be trapped:

- | | |
|-------|---------------------------|
| 1-10 | Function keys 1 to 10 |
| 11 | Up arrow |
| 12 | Left arrow |
| 13 | Right arrow |
| 14 | Down arrow |
| 15-20 | Keys defined by the form: |

KEY (n),CHR\$(KBflag) + CHR\$ (scan code)

Keys 15-20 can be trapped only in VBASICA 2.0 and later releases. See the KEY (n) statement for more information.

<line number> is a valid number from 0 to 65529. If <line number> is 0, VBASICA disables trapping on the specified key.

Execute a KEY(<n>) ON statement to activate trapping of function key or cursor control key activity. After KEY(<n>) ON, if you specify a nonzero line number in the ON KEY(<n>) statement, every time VBASICA starts a new statement it checks if the specified key was pressed. If the key was pressed, VBASICA performs a GOSUB to the line number specified in the ON KEY(<n>) statement.

If you execute a `KEY(<n>) OFF` statement, no trapping occurs for the specified key and the event is not remembered even if it does occur.

If you execute a `KEY(<n>) STOP` statement, no trapping occurs. However, if the specified key is pressed, VBASICA remembers this event. Consequently, an immediate trap occurs when you execute a `KEY(<n>) ON`.

When the trap occurs, VBASICA executes an automatic `KEY(<n>) STOP`. Thus, recursive traps can never take place. The `RETURN` from the trap routine automatically performs a `KEY(<n>) ON` unless an explicit `KEY(<n>) OFF` was performed inside the trap routine.

Event trapping does not occur when VBASICA is not executing a program. When an error trap (resulting from an `ON ERROR` statement) occurs, VBASICA automatically disables all trapping (including `ERROR`, `COM`, and `KEY`).

Key trapping may not work when you press other keys before the specified key. The key that caused the trap cannot be tested using `INPUT$` or `INKEY$`. Therefore, the trap routine for each key must be different if you desire a different function.

`KEY(<n>) ON` has no effect on whether the soft key values are displayed on the 25th line.

See the `RETURN` statement for more information.

EXAMPLE:

```

100 KEY.5 = 5
110 ON KEY(KEY.5) GOSUB 500
    .
    .
    .
500 '*****      ROUTINE TO HANDLE KEY(5)
    .
    .
    .
550 RETURN

```

ON PLAY Statement

FORMAT:

ON PLAY (n) GOSUB <linenumber>

PURPOSE:

Branches to a specified subroutine when the music queue contains fewer than (n) notes. This statement permits continuous music during program execution.

REMARKS:

(n) is an integer expression from 1 through 32. Values outside this range result in an "Illegal function call" error.

<linenumber> is the statement line number of the PLAY event trap subroutine.

PLAY ON causes an event trap when the background music queue goes from (n) notes to (n - 1) notes.

(n) must be an integer between 1 and 255.

PLAY ON enables PLAY event trapping.

PLAY OFF disables PLAY event trapping.

PLAY STOP suspends PLAY event trapping.

If you execute a PLAY OFF statement, VBASICA does not perform or remember the GOSUB.

If you execute a PLAY STOP statement, VBASICA does not perform the GOSUB until it executes a PLAY ON statement.

When an event trap occurs (that is, the GOSUB is performed), VBASICA executes an automatic PLAY STOP so that recursive traps cannot occur. The RETURN from the trapping subroutine automatically performs a PLAY ON statement unless an explicit PLAY OFF occurred inside the subroutine.

You can use the RETURN <line number> form of the RETURN statement to return to a specific line number from the trapping subroutine. Use this type of return with care because any other GOSUBs, WHILEs, or FORs active at the time of the trap remain active, and errors such as "FOR without NEXT" might result.

RULES:

1. A PLAY event trap is issued only when playing background music (for example, PLAY "MB..). PLAY event traps are not issued when running in Music Foreground (for example, default case, or PLAY "MF..).
2. A PLAY event trap is not issued if the background music queue has already gone from having (n) to (n - 1) notes when a PLAY ON is executed.
3. If (n) is a large number, event traps occur frequently enough to diminish program execution speed.

Also see the PLAY ON, PLAY OFF, and PLAY STOP statements.

EXAMPLE:

In the following example, control branches to a subroutine when the background music buffer decreases to 7 notes.

```
100 PLAY ON
.
.
.
540 PLAY "MB L1 XZITHER$"
550 ON PLAY(8) GOSUB 6000
.
.
.
6000 REM, **BACKGROUND MUSIC**
6010 LET COUNT% = COUNT% + 1
.
.
.
6999 RETURN
```

3

ON TIMER Statement

FORMAT:

ON TIMER (n) GOSUB <line number>

PURPOSE:

Provides an event trap during real time.

REMARKS:

ON TIMER causes an event trap every (n) seconds. (n) must be a numeric expression from 1 to 86400 (1 second to 24 hours). Values outside this range generate an "Illegal function call" error.

The ON TIMER statement is executed only if a TIMER ON statement is executed to enable event trapping. If event trapping is enabled and the <line number> in the ON TIMER statement is not zero, VBASICA checks between statements to see if the time has been reached. If it has, a GOSUB is performed to the specified line.

If a TIMER OFF statement has been executed, the GOSUB is not performed and is not remembered.

If a TIMER STOP statement has been executed, the GOSUB is not performed, but will be performed as soon as a TIMER ON statement is executed.

When an event trap occurs (that is, the GOSUB is performed), an automatic TIMER STOP is executed so that recursive traps cannot occur. The RETURN from the trapping subroutine automatically performs a TIMER ON statement unless an explicit TIMER OFF was performed inside the subroutine.

You can use the RETURN <line number> form of the RETURN statement to return to a specific line number from the trapping subroutine. Use this type of return with care because any other GOSUBs, WHILEs, or FORs active at the time of the trap remain active, and errors such as “FOR without NEXT” can result.

EXAMPLE:

The following example displays the time of day on line 1 every minute.

```

10 ON TIMER(60) GOSUB 10000
20 TIMER ON

.
.
.

10000 LET OLDROW=CSRLIN 'Save current Row
10010 LET OLDCOL=POS(0)'Save current Column
10020 LOCATE 1,1:PRINT TIMES$;
10030 LOCATE OLDROW,OLDCOL 'Restore Row & Col
10040 RETURN

```

Also see the **TIMER ON**, **TIMER OFF**, and **TIMER STOP** statements.

OPEN Statement

FORMAT:

OPEN [**<dev>**] **<filespec>** [**FOR** **<mode>**] **AS** [**#**]
<file number> [**LEN =** **<lrecl>**]

PURPOSE:

Establishes addressability between a physical device and an I/O buffer in the data pool.

REMARKS:

<dev> is an optional part of the filename string.

<filespec> is a valid string expression for the file specification. In VBASICA 2.0 and later versions, it can contain a path. Refer to Chapter 1.5 for more information. In the simplest case, it is a filename. The device may be specified separately in the **<dev>** field or as part of the **<filespec>**, or omitted entirely, in which case the default drive is assumed.

< mode > determines the initial positioning within the file and the action to be taken if the file does not exist. The valid modes and actions taken are the following:

- | | |
|--------|--|
| INPUT | Position to the beginning of an existing file. The “File Not Found” error is given if the file does not exist. |
| OUTPUT | Position to the beginning of the file. If the file does not exist, one is created. |
| APPEND | Position to the end of the file. If the file does not exist, one is created. |

If you omit the FOR < mode > clause, the initial position is at the beginning of the file. If VBASICA does not find the file, it creates one in the Random I/O mode. Records can be read or written randomly at any position within the file.

< file number > is an integer expression returning a number from 1 through 255. Use this number to associate an I/O buffer with a disk file or device. This association exists until you execute a CLOSE < file number > or CLOSE statement.

NOTE: At any time, you can have a particular file open under more than one file number. Therefore, you can use different modes for different purposes. Or, for program clarity, you can use different file numbers for different modes of access. Each file number has a different buffer, so you can keep several records from the same file in memory for quick access. However, you cannot open a file for sequential output or append if the file is already open.

< lrecl > is an integer expression from 2 to 32768. This value sets the record length used for random files; see the FIELD statement. If omitted, the record length defaults to 128-byte records.

When you OPEN FOR APPEND a disk file, the position is initially at the end of the file and the record number is set to the last record of the file. Then GET#(<lrecl>), LOF(<lrecl>), PRINT, WRITE, or PUT extend the file. The program can position elsewhere in the file with a GET statement. If this procedure is done, the mode is changed to random and the position moves to the record indicated.

After VBASICA moves the position from the end of the file, you can append records to the file by executing a GET #x,LOF(x)/<lrecl>.

OPEN COM Statement

FORMAT:

OPEN <COM.specification> AS [#]<file number>

PURPOSE:

Allocates a buffer for I/O as OPEN for disk files.

REMARKS:

<COM.specification> is:

"<dev>:<speed>,<parity>,<data>,<stop>[,RS]
[,CS<n>][,DS<n>][,CD<n>][,LF][,PE][,ASC or ,BIN]"

<dev> is a valid communications device. Valid devices are COM1: (port A) and COM2: (port B).

< speed > is a literal integer specifying the transmit/receive baud rate. Valid speeds are the following:

50, 75, 110, 150, 200, 300, 600, 1200, 1800, 2000, 2400, 3600, 4800, 9600, 19200

< parity > is a one-character literal specifying the parity for transmit and receive as follows:

- | | | |
|---|-------|--|
| S | SPACE | Parity bit always transmitted and received as space (0 bit). |
| O | ODD | Odd transmit/receive parity checking. |
| M | MARK | Parity bit always transmitted and received as mark (1 bit). |
| E | EVEN | Even transmit/receive parity checking. |
| N | NONE | No transmit parity, no receive parity checking. |

< data > is a literal integer indicating the number of transmit/receive data bits. The following are valid values:

5, 6, 7, 8

NOTE: 5 data bits with no parity is illegal. 8 data bits with any parity is illegal.

< stop > is a literal integer indicating the number of stop bits. The following are valid values:

1, 2

If omitted, then 75 and 110 bps transmit two stop bits; all others transmit one stop bit.

[RS] suppresses RTS (Request to Send).

[CS < n >] controls CTS (Clear to Send).

[DS < n >] controls DSR (Data Set Ready).

[CD < n >] controls CD (Carrier Detect).

3

[LF] sends a line feed character after each carriage return, including the carriage return sent as a result of the width setting. LF allows communications files to be printed on a serial line printer. Note that INPUT# and LINE INPUT# stop when they encounter a carriage return, ignoring the line feed, when used to read from a communications file opened with the LF option.

[PE] enables parity checking. The default is no parity checking. The PE option causes a device I/O error on parity errors and turns on the high order bit for 7 or less data bits. The PE option does not affect framing and overrun errors. These errors always turn on the high order bit and cause a device I/O error.

< file number > is an integer expression returning a valid file number. The number is associated with the file for as long as it is OPEN and refers other COM I/O statements to the file.

DEFAULTS: Missing parameters invoke the following defaults: speed = 300 bps, parity = EVEN, bits = 7.

NOTE: You can OPEN a COM device to only one file number at a time.

POSSIBLE ERRORS:

Any coding errors within the filename string result in the "Bad File Name" error. VBASICA gives no indication of the parameter in error.

The "Device Timeout" error occurs if VBASICA does not detect data set ready (DSR). Refer to hardware documentation or the *DOS 2.1 Reference* for proper cabling instructions.

EXAMPLE:

VBASICA opens file 1 for communication with default values. Speed at 300 bps, even parity, 7 data bits, and 1 stop bit:

```
10 OPEN "COM1: " AS 1
```

VBASICA opens file 2 for communication at 2400 bps. Parity and number of data bits are the default values:

```
20 OPEN "COM1:2400 " AS #2
```

VBASICA opens file number 1 for Asynchronous I/O at 1200 bits/second, with no parity produced or checked, and sends and receives 8-bit bytes:

```
10 OPEN "COM1:1200,N,8" AS #1
```

OPTION BASE Statement

FORMAT:

OPTION BASE n

PURPOSE:

Declares the minimum value for array subscripts.

REMARKS:

n is 1 or 0.

The default base is 0.

EXAMPLE:

If the following statement is executed, the lowest value an array subscript can have is 1:

OPTION BASE 1

OUT Statement

FORMAT:

OUT I,J

PURPOSE:

Sends a byte to an output port.

REMARKS:

I and J are integer expressions in the range 0 to 65535.

The integer expression I is the port number, and the integer expression J is the data to be transmitted.

EXAMPLE:

This command:

```
100 OUT 12345,255
```

is the same as the following (in assembly language):

```
MOV DX,12345
MOV AL,255
OUT DX,AL
```

PAINT Statement

FORMAT:

**PAINT (<xstart> , <ystart>) [, <paint attribute>
[, <border attribute>]]**

PURPOSE:

Fills an area on the screen with the selected color, for Graphics mode only.

REMARKS:

<xstart> and <ystart> are valid coordinates that specify the screen coordinates for the origin of painting.

<paint attribute> is the color VBASICA paints specified by a number from 0 to 3. It determines which color fills the area.

<border attribute> is the color of the edges of the figure VBASICA paints specified by a number from 0 to 3. Painting continues until VBASICA finds this color.

The PAINT statement fills in an arbitrary graphics figure with the specified paint color. If not specified, the paint attribute defaults to the foreground color (3 or 1), and the border attribute defaults to the paint attribute.

For example, you might want to fill in a circle of attribute 3 with attribute 0 (a black ball with a white border).

Only two attributes exist in high-resolution mode: whitening out an area until white is encountered, or blacking out an area until black is encountered.

PAINT must start on a nonborder point or it has no effect.

PAINT can fill any figure, but PAINTing jagged edges or very complex figures can result in the “Out of Memory” error. If you get this error, increase the amount of stack space available with the CLEAR statement.

VBASICA clips out-of-range coordinates.

EXAMPLE:

```
10 SCREEN 2
20 LINE (100,200)-(200,350),1,B
30 PAINT (150,225),1,1
```

3

Tiling

Tiling is available for VBASICA 2.0 and later releases. It is the design of a PAINT pattern 8 bits wide and up to 64 bytes long. Each byte in the tile string masks 8 bits along the x-axis when putting down points. Construction of the tile mask works as follows. Use the syntax:

PAINT (x,y), CHR\$(n)...CHR\$(n)

where (n) is a number between 0 and 255 which will be represented in binary across the x-axis of the tile. Each CHR\$(n) up to 64 generates an image of the bit arrangement of the code for that character. For example, the decimal number 85 is binary 01010101. The graphic image line on a black and white screen generated by CHR\$(85) is an eight-pixel line, with even numbered pixels white, and odd pixels black. That is, each bit containing a 1 sets the associated pixel on and each bit filled with a 0 sets the associated bit off in monochrome mode. The ASCII character CHR\$(85), U, is not displayed in this case.

If the current screen mode supports only two colors, the screen can be painted with X's with the following statement:

```
PAINT (320,100),CHR$(129) + CHR$(66) + CHR$(36) + CHR$(24) +  
CHR$(24) + CHR$(36) + CHR$(66 + CHR$(129)
```

This appears on the screen as:

x increases →

0,0	x							x	CHR\$(129)	Tile byte 1
0,1		x						x	CHR\$(66)	Tile byte 2
0,2			x				x		CHR\$(36)	Tile byte 3
0,3				x	x				CHR\$(24)	Tile byte 4
0,4				x	x				CHR\$(24)	Tile byte 5
0,5			x				x		CHR\$(36)	Tile byte 6
0,6		x						x	CHR\$(66)	Tile byte 7
0,7	x							x	CHR\$(129)	Tile byte 8

When supplied, < background attribute > specifies the background tile slice to skip when checking for boundary termination.

You cannot specify more than two consecutive bytes in the tile background slice that match the tile string. Specifying more than two results in an "Illegal function call" error.

EXAMPLE:

```
10 PAINT (5,15),2,0
```

begins painting at coordinates 5,15 with color 2 and border color 0, and fills to a border.

PLAY Statement

FORMAT:

PLAY <string exp>

PURPOSE:

Plays music as specified by <string exp> .

REMARKS:

<string exp> is a string expression returning a valid string conforming to the format described in Table 3-5.

PLAY implements a concept similar to DRAW by embedding a Music Macro Language into the string data type. The following table describes the single-character commands you can use with the PLAY statement.

Table 3-5: PLAY Commands

COMMAND	DESCRIPTION
A-G [# , + , -]	Plays the note. A # or + following the note indicates sharp, and - indicates flat.
L <n>	Length sets the length of each note. L4 is a quarter note, L1 is a whole note, and so on. n ranges from 1 to 64. The length can also follow the note when you want to change the length for only one note. In this case, A16 is equivalent to L16A.
MF	Music Foreground. Music created by PLAY or SOUND runs in the foreground. That is, VBASICA does not execute the next program statement until the last note, or rest, of subsequent PLAY statements are started.
MB	Music Background. Music created by SOUND or PLAY runs in the background. That is, each note or sound is placed in a buffer, allowing the VBASICA program to continue executing while music plays in the background. Up to 32 notes (or rests) can be played in the background at a time.

COMMAND	DESCRIPTION
MN	Music Normal. Each note plays $\frac{1}{8}$ ths of the time determined by L (length).
ML	Music Legato. Each note plays the full period set by L (length).
MS	Music Staccato. Each note plays $\frac{3}{4}$ ths of the time determined by L (length).
N < n >	Play note n. n can range from 0 to 84. In the seven possible octaves, 84 notes exist. N = 0 indicates rest.
> < n >	Go to the next higher octave and play note n, for VBASICA 2.0 and later. Each time note n is played, the octave increases until it reaches octave 6. For example, PLAY "> A" raises the octave and plays note A. Each time PLAY "> A" is executed, the octave goes up until it reaches octave 6; then each time PLAY "> A" executes, note A plays at octave 6.
< < n >	Go to the next lower n and play note n, for VBASICA 2.0 and later. Each time note n is played, the octave decreases, until it reaches octave 0. For example, PLAY "< A" lowers the octave and plays note A. Each time PLAY "< A" executes, the octave decreases until it reaches octave 0; then each time PLAY "< A" executes, note A plays at octave 0.
O < n >	Octave. Sets the current octave. Seven octaves (0..6) exist.
P < n >	Pause. P ranges from 1 to 64.
T < n >	Tempo. Sets the number of L4's in a second. n ranges from 32 to 255. The default is 120. Period. A dot after a note plays the note $\frac{3}{2}$ times the period determined by L (length) times T (tempo). Multiple dots can appear after the note. The period is scaled accordingly (for example, A. $\frac{3}{2}$, A. . $\frac{9}{4}$, A. . . $\frac{27}{8}$). Dots can appear after a pause (P) and scale the pause length as described.
X < string >	Executes substring.

NOTE: Because of the slow clock interrupt rate, some notes do not play at higher tempos—for example, L64 at T255. Determine these note/tempo combinations through experimentation. In all forms of the PLAY command, the n argument can be a constant such as 12 or it can be the name of a variable.

To play tied notes, concatenate the expressions of the two notes. VBASICA plays the two notes continuously.

You can use X to store a “subtune” in one string and call it repetitively with different tempos or octaves from another string.

EXAMPLE:

```
10 A$ = "BB-C"  
20 B$ = "O4XA$;"  
30 C$ = "L1CT5ON3N4N5N6"  
40 PLAY "P2XA$;XB$;XC$;"
```

3

PLAY(n) Function

FORMAT:

PLAY (n)

PURPOSE:

Returns the number of notes currently in the background music queue.

REMARKS:

(n) is a dummy argument and can be any value.

PLAY(n) returns 0 when you are in Music Foreground mode.

PLAY ON, PLAY OFF, and PLAY STOP Statements

FORMAT:

PLAY ON

PLAY OFF

PLAY STOP

3

PURPOSE:

PLAY ON enables **PLAY** event trapping, specified by the **ON PLAY** statement.

PLAY OFF disables **PLAY** event trapping.

PLAY STOP suspends **PLAY** event trapping.

REMARKS:

If a **PLAY OFF** statement was executed, the **GOSUB** is not performed and is not remembered.

If a **PLAY STOP** statement was executed, the **GOSUB** is not performed, but is performed as soon as a **PLAY ON** statement is executed.

When an event trap occurs (that is, the **GOSUB** is performed), **VBASICA** executes an automatic **PLAY STOP** so recursive traps cannot occur. The **RETURN** from the trapping subroutine automatically performs a **PLAY ON** statement unless an explicit **PLAY OFF** was performed inside the subroutine.

You can use the **RETURN <line number>** form of the **RETURN** statement to return to a specific line number from the trapping subroutine.

NOTE: Any other GOSUBs, WHILEs, or FORs active at the time of the trap remain active, and errors such as “FOR without NEXT” can result.

PMAP Function

FORMAT:

PMAP < expression > , < function >

3

PURPOSE:

Maps world coordinate expressions to physical locations or maps physical expressions to a world coordinate location for graphics mode.

< function > can be:

- 0 Maps world expression to physical x coordinate.
- 1 Maps world expression to physical y coordinate.
- 2 Maps physical expression to world x coordinate.
- 3 Maps physical expression to world y coordinate.

REMARKS:

The four PMAP functions allow you to find equivalent point locations between the world coordinates created with the WINDOW statement and the physical coordinate system of the screen or viewport as defined by the VIEW statement.

EXAMPLE:

If you define a WINDOW SCREEN (80,100)–(200,200) then the upper left coordinate of the window is (80,100) and the lower right is (200,200). The range of the screen coordinates can be (0,0) in the upper left corner and (639,199) in the lower right. Then,

`X = PMAP(80,0)`

returns the screen x coordinate of the window x coordinate 80:

0

The PMAP function in the statement:

`Y = PMAP(200,1)`

returns the screen y coordinate of the window y coordinate 200:

199

The PMAP function in the statement:

`X = PMAP(619,2)`

returns the “world” x coordinate that corresponds to the screen or viewport x coordinate 619:

199

The PMAP function in the statement:

`Y = PMAP(100,3)`

returns the “world” y coordinate that corresponds to the screen or viewport y coordinate 100:

140

POINT Function

FORMAT:

POINT (< xcoordinate > , < ycoordinate >)

or

POINT (< function >)

PURPOSE:

POINT (x,y) allows you to read the color number of a pixel from the screen. If the specified point is out of range, the value - 1 is returned. In medium-resolution graphics, valid returns are 0 through 3, and in high-resolution, 0 through 1.

REMARKS:

< xcoordinate > and < ycoordinate > are the coordinates of the pixel to be referenced, for Graphics mode only.

POINT with one argument allows you to retrieve the current graphics cursor coordinates. Therefore:

a = POINT < function >

returns the value of the current x or y graphics accumulator, depending on the value of < function > , as follows:

< function > = Action of POINT < function >

- 0 Returns the current physical x coordinate.
- 1 Returns the current physical y coordinate.
- 2 Returns the current logical x coordinate. If the WINDOW statement was not used, this returns the same value as the POINT(0) function.

- 3 Returns the current logical y coordinate if WINDOW is active, or else returns the current physical y coordinate, as in 1.

where the physical coordinate is the coordinate on the screen or current viewport.

EXAMPLE:

```
10 FOR I = 1 TO 400
20   IF POINT(I,I) <> 0 THEN GOTO 50   'a dot ?
30   PSET(I,I)      'put a dot if not one here
40   GOTO 60
50   PRESET(I,I)    'remove a dot if one here
60 NEXT I
```

POKE Statement

FORMAT:

POKE I,J

PURPOSE:

Writes a byte into a memory location.

REMARKS:

I and J are integer expressions.

I is the address of the memory location to be POKEd. I must be from 0 to 65536. J is the data to be POKEd. J must be from 0 to 255.

PEEK is the complementary function to POKE. The argument to PEEK is an address from which a byte is read.

EXAMPLE:

```
10 POKE &H5A00,&HFF
```

POS Function

FORMAT:

POS(X)

3

PURPOSE:

Returns the current cursor position. The leftmost position is 1. X is a dummy argument.

See also the LPOS function.

EXAMPLE:

```
IF POS(X)>60 THEN PRINT CHR$(13)
```

PRESET Statement

FORMAT:

PRESET (<absolute x> , <absolute y>) [, <attribute>]

PRESET STEP (<x offset> , <y offset>) [, <attribute>]

PURPOSE:

Removes (turns off) or displays (turns on) one pixel from the screen, for Graphics mode only.

REMARKS:

PRESET has a syntax identical to PSET. The only difference is that if no third parameter is given for the background color, zero is selected. When you give a third argument, PRESET is identical to PSET.

If an out-of-range coordinate is given to PSET or PRESET, no action is taken, nor is an error given. If an attribute greater than 3 is given, the "Illegal function call" error results. Attribute value 2 is treated like 0 in high resolution and 3 is treated like 1 for compatibility with medium resolution.

VBASICA clips out-of-range coordinates.

EXAMPLE:

```
10 FOR I = 0 TO 100
20     PSET (I,I)
30 NEXT     'draw a diagonal line to (100,100)
40 FOR I = 100 TO 0 STEP -1
50 PRESET (I,I),0
60 NEXT     'remove the line just drawn
```

PRINT Statement

FORMAT:

PRINT [<expr list>]

PURPOSE:

Outputs data to the screen.

REMARKS:

If you omit <expr list>, VBASICA prints a blank line. If you include <expr list>, the values of the expressions appear on the screen. The expressions in the list can be numeric and/or string expressions. Strings must be enclosed by quotation marks.

3

Print Positions

VBASICA divides each line into print zones of 14 spaces. The punctuation that separates the items in the list determines the position of each printed item. A comma prints the next value at the beginning of the next zone. A semicolon prints the next value immediately after the last value. One or more spaces between expressions is equivalent to typing a semicolon.

If <expr list> ends with a comma or a semicolon, the next PRINT begins printing on the same line, spacing accordingly. If the <expr list> terminates without a comma or a semicolon, VBASICA prints a Return at the end of the line. If the printed line exceeds the screen width, VBASICA continues printing on the next physical line.

A space always follows printed numbers. A space precedes positive numbers. A minus sign precedes negative numbers.

If VBASICA can represent a single precision number with six or fewer digits in the unscaled format as accurately as represented in the scaled format, VBASICA outputs that number in the unscaled format. For example, $1\text{E}-7$ is output as .0000001 and $1\text{E}-8$ is output as $1\text{E}-08$. If a double precision number can be represented with 16 or fewer digits in the unscaled format as accurately as represented in the scaled format, VBASICA outputs the number in the unscaled format. For example, $1\text{D}-16$ is output as .00000000000000001 and $1\text{D}-17$ is output as $1\text{D}-17$.

EXAMPLE:

In the following example, the commas in the PRINT statement print each value at the beginning of the next print zone.

```
10 X=5
20 PRINT X+5,X-5,X*(-5),X^5
30 END
RUN
10      0      -25      3125
Ok
```

In the following example, the semicolon at the end of line 20 prints both PRINT statements on the same line. Line 40 prints a blank line before the next prompt.

```
LIST
10 INPUT X
20 PRINT X "SQUARED IS" X^2 "AND";
30 PRINT X "CUBED IS" X^3
40 PRINT
50 GOTO 10
Ok
RUN
?9
9 SQUARED IS 81 AND 9 CUBED IS 729

?21
21 SQUARED IS 441 AND 21 CUBED IS 9261

?
```

In the following example, the semicolons in the PRINT statement print each value immediately after the preceding value. A space always follows a number, and a space precedes positive numbers.

```
10 FOR X = 1 TO 5
20 J=J+5
30 K=K+10
40 PRINT J;K;
50 NEXT X
Ok
RUN
5 10 10 20 15 30 20 40 25 50
Ok
```

3

PRINT USING Statement

FORMAT:

PRINT USING <str expr>;<expr list>

PURPOSE:

Prints strings or numbers in specified format.

REMARKS:

<expr list> is a list of string or numeric expressions to be printed, separated by semicolons.

<str expr> is a string literal or variable consisting of special formatting characters.

<expr list> and <str expr> determine the field size and the format of the printed strings.

String Fields

When PRINT USING is used to print strings, one of three formatting characters can be used to format the string field:

- ▶ The ! specifies that only the first character of the given string is printed.
- ▶ \n spaces\ . Two characters from the string are to be printed. If you put spaces between the backslashes, each space adds another character to the printed string. For example, if you use one space, three characters are printed.
- ▶ If the string to be printed is longer than the field, VBASICA ignores the extra characters. If the field is longer than the string, the string is left-justified in the field and padded with spaces on the right. For example,

```
10 A$="LOOK":B$="OUT"
30 PRINT USING "!",A$;B$
40 PRINT USING "\ \ ";A$;B$
50 PRINT USING "\ \ ";A$;B$;"!!"
RUN
LO
LOOKOUT
LOOK OUT !!
```

- ▶ The & specifies a variable-length string field. When you specify the field with the optional &, the string is output exactly as input. For example,

```
10 A$="LOOK":B$="OUT"
20 PRINT USING "!",A$;
30 PRINT USING "&";B$
RUN
LOUT
```

Numeric Fields

When you print numbers with `PRINT USING`, VBASICA uses the following special characters to format the numeric field:

- ▶ A number sign (#) represents each digit position. Digit positions are always filled. If the number to be printed has fewer digits than the number of positions specified, the number is right-justified (preceded by spaces) in the field.
- ▶ You can insert a decimal point at any position in the field. If the format string specifies that a digit is to precede the decimal point, VBASICA always prints the digit (as 0 if necessary). VBASICA rounds numbers as necessary.

```
PRINT USING "##.##"; .78
0.78
```

```
PRINT USING "###.##"; 987.654
987.65
```

In the following example, the three spaces inserted at the end of the format string separate the printed values on the line:

```
PRINT USING "##.##    "; 10.2, 5.3, 66.789, .234
10.20    5.30    66.79    0.23
```

- ▶ A plus sign (+) at the beginning or end of the format string prints the sign of the number (plus or minus) before or after the number:

```
PRINT USING "+##.##"; -68.95, 2.4, 55.6, -9
-68.95          +2.40    +55.60          -0.90
```

- ▶ A minus sign (−) at the end of the format field prints negative numbers with a trailing minus sign, as in this example:

```
PRINT USING "##.##-"; -68.95, 22.44900, -7.01
68.95-    22.45    7.01-
```

- ▶ A double asterisk (**) at the beginning of the format string fills the leading spaces in the numeric field with asterisks. The ** also specifies positions for two more digits. For example,

```
PRINT USING "***#.##"; 12.39, -0.9, 765.1
*12.4      *-0.9      765.1
```

- ▶ A double dollar sign (\$\$) specifies two digit positions; one is the dollar sign. The \$\$ prints a dollar sign immediately to the left of the formatted number. Do not use the exponential format with \$\$\$. Do not use negative numbers unless the minus sign trails to the right. For example,

```
PRINT USING "$$###.##"; 456.78
$456.78
```

- ▶ The **\$ specifies three digit positions, one of which is the dollar sign. A **\$ at the beginning of a format string combines the effect of the preceding two symbols. Leading spaces are asterisk-filled. A dollar sign is printed before the number.

```
PRINT USING "**$###.##"; 2.34
**$2.34
```

- ▶ A comma specifies another digit position. A comma to the left of the decimal point in a formatting string prints a comma to the left of each third digit left of the decimal point. VBASICA prints a comma at the end of the format string as part of the string. The comma has no effect if used with the exponential format. For example,

```
PRINT USING "###, .###"; 1234.5
1,234.50

PRINT USING "#####.##, "; 1234.5
1234.50,
```

- Four carets (~~~~) specify exponential format when located after digit-position characters. The four carets allow space for E, a plus sign, and two digits to be printed. Any decimal point position can be specified. The significant digits are left-justified, and the exponent is adjusted. Unless you specify a leading or trailing plus (or minus) sign, one digit position to the left of the decimal point is used to print a space or a minus sign.

```
PRINT USING "###.###"; ^^^^234.56
2.35E+02
```

```
PRINT USING ".#####^--"; 888888
.8889E+06
```

```
PRINT USING "+.###^"; 123
.12E+03
```

- An underscore (_) causes the next character to be output as a literal character:

```
PRINT USING "_!##.##_!"; 12.34
!12.34!
```

To print an underscore, put two underscore characters into the format string.

- A percent sign (%) prints a percent sign before a number if that number is larger than the specified numeric field. If rounding causes the number to exceed the field, a percent sign is printed in front of the rounded number.

```
PRINT USING "###.###"; 111.22
%111.22
```

```
PRINT USING ".###"; .999
%1.00
```

If the number of digits specified exceeds 24, an “Illegal function call” error results.

PRINT# and PRINT# USING Statements

FORMAT:

PRINT# < filenum > [, [USING < str expr > ;] < expr list >

PURPOSE:

Writes data to a sequential disk file.

REMARKS:

< filenum > is the number used when the file was OPENed for output.

< str expr > consists of the formatting characters used with PRINT USING.

< expr list > is the numeric and/or string expressions written to the file.

PRINT# does not compress data on the disk. The data is written to the disk in the same form that VBASICA displays it on the screen by a PRINT statement. Delimit the data so it is input correctly to the disk.

Separate numeric expressions in < expr list > by semicolons. For example,

```
PRINT#1, A; B; C; X; Y; Z
```

If you use commas as delimiters, the extra blanks inserted between print fields are also written to disk.

Separate string expressions in < expr list > with semicolons. Use explicit delimiters to format the string expressions correctly on the disk.

For example, assume that A\$ is "CAMERA" and B\$ is "93604-1".

The statement:

```
PRINT#1,A$;B$
```

writes CAMERA 93604-1 onto disk. Because there are no delimiters, this line cannot be input as two separate strings. To correct the problem, insert delimiters into the PRINT# statement as shown:

```
PRINT#1,A$;" ";B$
```

CAMERA,93604-1 is written to disk. In the new format, the data can be read back into two string variables.

If the strings contain commas, semicolons, significant leading blanks, Returns, or linefeeds, enclose each with explicit quotation marks by using CHR\$(34). Assume that A\$ is "CAMERA, AUTOMATIC" and B\$ is "93604-1".

The statement:

```
PRINT#1,CHR$(34);A$,CHR$(34);CHR$(34);B$;CHR$(34)
```

writes the following to disk:

```
"CAMERA, AUTOMATIC" "93604-1"
```

and the statement:

```
INPUT#1,A$,B$
```

inputs "CAMERA, AUTOMATIC" to A\$ and "93604-1" to B\$.

You can also use the PRINT# statement with the USING option to control the format of the disk file, as in this example:

```
PRINT#1,USING"$$$###.##";J;K;L
```

See Appendix G for more information on disk file formatting.

PSET Statement

FORMAT:

PSET (<absolute x>,<absolute y>) [,<attribute>]

PSET STEP (<x offset>,<y offset>) [,<attribute>]

PURPOSE:

Displays (turns on) or removes (turns off) one pixel from the screen, for Graphics mode only.

REMARKS:

<absolute x>, <absolute y>, <x offset>, and <y offset> are valid coordinates. See Chapter 1.1 for further information.

<attribute> is an expression returning a value 0 to 3, which determines the color of the point. An attribute of 0 sets the point to the background color, as described in Chapter 1.1.

If you omit the attribute argument, it defaults to 3 in Screen mode 1 and to a value of 1 in Screen mode 2.

VBASICA clips out-of-range coordinates.

EXAMPLE:

```
10 FOR I = 0 TO 100
20 PSET (I,I)
30 NEXT '(draw a diagonal line to (100,100))
40 FOR I = 100 TO 0 STEP -1
50 PSET (I,I),0
60 NEXT '(remove the line just drawn)
```

RANDOMIZE Statement

FORMAT:

RANDOMIZE [< expr >]

PURPOSE:

Reseeds the random number generator.

REMARKS:

< expr > is an integer, single or double precision expression used as the random number seed. If you omit < expr > , VBASICA suspends program execution and asks for a value. The following prompt appears on the screen:

Random Number Seed (-32768 to 32767)?

RANDOMIZE executes after you supply a value.

If the random number generator is not reseeded, the RND function returns the same sequence of random numbers each time the program is run. To change the sequence of random numbers, put a RANDOMIZE statement at the beginning of the program and change its argument each time you run the program.

EXAMPLE:

```
10 RANDOMIZE
20 FOR I=1 TO 5
30 PRINT RND;
40 NEXT I
```

RUN

Random Number Seed (-32768 to 32767) ? 3

.88598 .484668 .586328 .119426 .709225

Ok

RUN

Random Number Seed (-32768 to 32767) 4

.803506 .162462 .929364 .292443 .322921

Ok

RUN

Random Number Seed (-32768 to 32767)? 3

.88598 .484668 .586328 .119426 .709225

Ok

READ Statement

FORMAT:

READ < varlist >

PURPOSE:

Reads values from a DATA statement and assigns them to variables.

REMARKS:

Always use a READ statement with a DATA statement. READ statements assign variables to DATA statement values, one for one. READ statement variables can be numeric or string, and the values read must agree with the variable types specified. If they do not agree, a “Syntax error” results.

A single READ statement can access one or more DATA statements (in order), or several READ statements can access the same DATA statement. If the number of variables in < varlist > exceeds the number of elements in the DATA statements, subsequent READ statements begin reading data at the first unread element. If no subsequent READ statements exist, VBASICA ignores the extra data.

Use the RESTORE statement to read DATA statements from the start.

EXAMPLE:

The following program segment READs the values from the DATA statements into the array A. After execution, the value of A(1) is 3.08, and so on.

```
.  
. .  
. .  
80 FOR I=1 TO 10  
90 READ A(I)  
100 NEXT I  
110 DATA 3.08,5.19,3.12,3.98,4.24  
120 DATA 5.08,5.55,4.00,3.16,3.37  
. .  
. .  
. .
```

The following program READs string and numeric data from the DATA statement in line 30:

```
LIST  
10 PRINT "CITY", "STATE", "ZIP"  
20 READ C$,S$,Z  
30 DATA "DENVER," , COLORADO, 80211  
40 PRINT C$,S$,Z  
Ok  
RUN  
CITY        STATE      ZIP  
DENVER,     COLORADO 80211  
Ok
```

REM Statement

FORMAT:

REM < remark >

PURPOSE:

Inserts explanatory remarks and comments into a program.

REMARKS:

VBASICA does not execute REM statements. They are output exactly as entered when the program is listed.

REM statements can be branched into from a GOTO or GOSUB statement. Execution continues at the first executable statement after the REM statement.

You can add remarks to the end of a line by preceding the remark with a single quotation mark.

WARNING: Do not use REM in a DATA statement. VBASICA treats it as data, not as a remark.

EXAMPLE:

```
.  
.   
120 REM CALCULATE AVERAGE VELOCITY  
130 FOR I=1 TO 20  
140 SUM=SUM +V(I)  
.   
.   
.   
120 FOR I=1 TO 20 'CALCULATE AVERAGE  
    VELOCITY  
130 SUM=SUM+V(I)  
140 NEXT I  
.   
.   
. 
```

3

RENUM Command

FORMAT:

RENUM [[< new number >],[< old number >], < increment >]]

PURPOSE:

Renumbers program lines.

REMARKS:

< new number > is the first line number in the new sequence. The default is 10.

<old number> is the line where renumbering is to begin. The default is the first line of the program.

<increment> is the increment used in the new sequence. The default is 10.

RENUM changes all line number references following GOTO, GOSUB, THEN, ON...GOTO, ON...GOSUB, and ERL statements to reflect the new line numbers. If a nonexistent line number appears after one of these statements, VBASICA prints the error message "undefined line xxxxx in yyyy". RENUM does not change the incorrect line number reference (xxxxx), but the line number (yyyy) can be changed.

You cannot use RENUM to change the order of program lines or to create line numbers greater than 65529. An "Illegal function call" error results in both cases.

EXAMPLES:

In the following example, the statement renumbers the entire program. The first new line number is 10. Lines increment by 10.

```
RENUM
```

In the following statement, VBASICA renumbers the entire program. The first new line number is 300. Lines increment by 50.

```
RENUM 300, ,50
```

In the following statement, VBASICA renumbers the lines beginning at 900. The new numbering begins with 1000. Each line increments by 20.

```
RENUM 1000,900,20
```

RESET Command

FORMAT:

RESET

PURPOSE:

Closes all files.

3

REMARKS:

RESET closes all open files and writes all blocks in memory to disk. Thus, if the machine loses power, VBASICA properly updates all files. You must close all files before you remove a disk from its drive.

EXAMPLE:

```
998 RESET
999 END
```

RESTORE Statement

FORMAT:

RESTORE [< line >]

PURPOSE:

Reads DATA statements, beginning at a specified line.

REMARKS:

< line > is the number of a program line.

After VBASICA executes a RESTORE statement, the next READ statement begins with the first item in the program's first DATA statement. If you specify < line >, the next READ statement starts at the first item in the specified DATA statement.

EXAMPLE:

```
10 READ A,B,C
20 RESTORE
30 READ D,E,F
40 DATA 57,68,79
```

.
.
.

RESUME Statement

FORMAT:

RESUME

RESUME 0

RESUME NEXT

RESUME < line >

3

PURPOSE:

Continues program execution after an error-recovery procedure.

REMARKS:

< line > is the number of a program line.

You can use any of the four formats shown, depending on where execution resumes:

- ▶ **RESUME** and **RESUME 0** resume execution at the statement that caused the error.
- ▶ **RESUME NEXT** resumes execution at the statement immediately following the one that caused the error.
- ▶ **RESUME < line >** resumes execution at the line specified.

A **RESUME** statement that appears outside of an error-trapping routine causes a “RESUME without error” message to appear.

EXAMPLE:

```
10 ON ERROR GOTO 900
.
.
900 IF (ERR=230)AND(ERL=90) THEN PRINT
    "TRY AGAIN":RESUME 80
.
.
```

RETURN Statement

FORMAT:

RETURN [< line number >]

PURPOSE:

Returns from a GOSUB.

REMARKS:

The line number is intended for use with event trapping. The event trap routine might want to go back into the VBASICA program at a fixed line number while still eliminating the GOSUB entry the trap created.

Use the nonlocal return with care. Any other GOSUB, WHILE, or FOR active at the time of the trap remains active. If the trap comes out of a subroutine, any attempt to continue loops outside the subroutine results in the "NEXT without FOR" error.

EXAMPLE:

```
100 RETURN 900
```

RMDIR Command

FORMAT:

RMDIR <pathname>

PURPOSE:

Removes an existing directory, for VBASICA 2.0 and later.

3

REMARKS:

<pathname> is the name of the directory to be deleted. RMDIR works exactly like the DOS command RMDIR. The <pathname> must be a string of less than 63 characters.

The <pathname> to be removed must be empty of any files except the working directory (.) and the parent directory (..) or else VBASICA gives a "Path not found" or a "Path/File access" error.

EXAMPLE:

In the following statement, the SALES directory on the current drive is removed:

RMDIR "\SALES"

RND Function

FORMAT:

RND[(X)]

PURPOSE:

Returns a random number between 0 and 1. VBASICA generates the same sequence of random numbers each time the program is RUN unless you reseed the random number generator.

If X is less than zero, RND always restarts the same sequence for any given X. If X is greater than zero, or X is omitted, RND generates the next random number in the sequence. If X equals zero, RND repeats the last number generated.

EXAMPLE:

The following example prints five random numbers between 0 and 100.

```
10 FOR I=1 TO 5
20 PRINT INT(RND*100);
30 NEXT I
```

RUN Command

FORMAT 1:

RUN [<line>]

FORMAT 2:

RUN <filespec> [,R]

PURPOSE:

Format 1 executes the program in memory. Format 2 loads a disk file into memory and runs it.

REMARKS:

<line> is the number of a program line.

With Format 1, execution begins at <line>. If you do not specify <line>, execution starts at the lowest line number. VBASICA returns to command level after it executes a RUN.

In Format 2, <filespec> is a string expression for the specification. In VBASICA 2.0 and later, it can contain a path. Refer to Chapter 1.5 for more information on file specifications.

RUN closes all open files and deletes the current contents of memory before loading the designated program. If you use the R option, all data files remain open.

EXAMPLE:

RUN "NEWFIL",R

VBASICA supports the RUN and RUN <line number> forms of the RUN statement; however, it does not support the R option. If you want this feature, use the CHAIN statement.

SAVE Command

FORMAT:

SAVE " < filespec > " [,A,P]

PURPOSE:

Saves a VBASICA program file on disk or another device.

REMARKS:

< filespec > is a string expression representing the file specification. In VBASICA 2.0 and later, it can contain a path. Refer to Chapter 1.5 for more information on file specifications. Device specifications other than the diskette are legal.

The A option saves the file in ASCII format. If the A option is not specified, VBASICA saves the file in a compressed binary format. ASCII format takes more space on the disk, but some actions require that files be in ASCII format. For example, the MERGE command requires an ASCII format file, and some operating system commands such as TYPE can require an ASCII format file.

The P option protects the file by saving it in an encoded binary format. When a protected file is later RUN (or LOADED), any attempt to list or edit it will fail.

EXAMPLE:

SAVE "COM2" , A

Saves the program COM2.BAS in ASCII format.

SAVE "PROG" , P

Saves the program PROG.BAS as a protected file which cannot be altered.

SCREEN Statement

FORMAT:

SCREEN [< mode >] [, [< burst >] [, [< apage >] [, < vpage >]]]

PURPOSE:

Sets the screen attributes.

REMARKS:

< mode > is a valid numeric expression returning an unsigned integer value. Valid modes are the following:

- | | |
|---------|--|
| 0 | Alpha mode at current width (40 or 80) |
| 1 and 2 | Graphics modes |

< burst > is a valid numeric expression returning a Boolean result. (Not currently in use.)

< apage > is the active page. This value can be nonzero only on Screen 0. In this mode, it is a numeric expression returning an unsigned integer from 0 to 7 for width 40, or 0 to 3 for width 80. VBASICA sends output to the screen to this page.

< vpage > is the visible page. It follows the same rules as < apage >, but selects the page to be displayed on the screen. If < vpage > is omitted, the visible page is set to the active page.

If all parameters are legal and < mode > or < burst > changes from their previous values, VBASICA clears the screen. In that case, the background color is reset to black, and the foreground color to white.

If the mode is 0, and you specify only <apage> and <vpage>, VBASICA changes display pages for viewing. Initially, both active and visual pages default to zero. By manipulating active and visual pages, you can display one page while constructing another. You can switch both pages instantaneously.

NOTE: Only one cursor is shared between the pages. If you are going to switch active pages back and forth, save the cursor position on the current page (using POS(0) and CSRLIN) before changing to another active page. When you return to the original page, you can return the cursor to the saved position using the LOCATE statement.

RULES:

1. Any values you enter outside these ranges result in the "Illegal function call" error. VBASICA retains the previous values.
2. You can omit any parameter. Omitted parameters assume the old value.

EXAMPLE:

```
10 SCREEN 0,0,0      'select alpha mode
10 SCREEN 2          'select hi-res graphics
```

SCREEN Function

FORMAT:

x = SCREEN (<row> , <col> [, <boolean>])

PURPOSE:

Returns the ASCII code value of the character from the screen at the specified row (line) and column.

3

ACTION:

x is a numeric variable receiving the ordinal returned.

<row> is a valid numeric expression returning an unsigned integer in the range 1 to 25.

<col> is a valid numeric expression returning an unsigned integer in the range 1 to 40 or 1 to 80, depending on the width.

<boolean> is a valid numeric value or expression giving a true or false (Boolean) result.

The ASCII code value of the character at the specified coordinates is stored in the numeric variable. If the optional parameter <boolean> is given and nonzero, the color attribute for the character is returned instead. The color attribute is a number in the range 0 to 255. This parameter may be interpreted as follows:

- ▶ (<param> MOD 16) is the foreground color. (For the standard screen and printer, values of 8 through 15 indicate high intensity for "colors" 0 through 7.)
- ▶ ((<param> MOD 128) – foreground) is the background color, where foreground is calculated as shown.
- ▶ (<param> > 127) is true (– 1) if the character is blinking, false (0) if not.

RULE: Any values entered outside these ranges result in the “Illegal function call” error.

EXAMPLE:

```
100 X = SCREEN (10,10)    'Returns 65 if the
                           'character at 10,10 is an 'A'.
110 X = SCREEN (1,1,1)    'Returns the color
                           'attribute of the character in the
                           'upper left corner of the screen.
```

3

SGN Function

FORMAT:

SGN(X)

PURPOSE:

Indicates the value of X, relative to zero:

- If $X > 0$, SGN(X) returns 1.
- If $X = 0$, SGN(X) returns 0.
- If $X < 0$, SGN(X) returns -1 .

EXAMPLE:

```
ON SGN(X)+2 GOTO 100,200,300
```

Branches to 100 if X is negative, 200 if X is 0, and 300 if X is positive.

SHELL Statement

FORMAT:

SHELL [< command-string >]

PURPOSE:

Exits the VBASICA program, runs a .COM or .EXE or .BAT program, or a built-in DOS function such as DIR or TYPE, and returns to the VBASICA program at the line after the SHELL statement.

REMARKS:

A .COM, .EXE, or .BAT program or DOS function that runs under the SHELL statement is called a child process. SHELL executes child processes by loading and running a copy of COMMAND with the /C switch. By using COMMAND in this way, command line parameters are passed to the child. Standard input and output can be redirected, and built-in commands such as DIR, PATH, and SORT can be executed.

The < command-string > must be a valid string expression containing the name of a program to run and optional command arguments.

The program name in < command-string > can have any extension. If you supply no extension, COMMAND looks for a .COM file, then an .EXE file, and finally, a .BAT file. If COMMAND is not found, SHELL issues a "File not found" error. VBASICA generates no error if COMMAND cannot find the file specified in < command-string > .

COMMAND processes any text separated from the program name by at least one blank as program parameters.

VBASICA remains in memory while the child process is running. When the child finishes, VBASICA continues.

WARNING: Do not attempt to SHELL to VBASICA as a child process. This option is not supported, and causes unpredictable results. Integrity of the parent VBASICA may or may not be harmed (in RAM only).

SHELL with no <command-string> gives you a new COMMAND shell. You can now do anything that COMMAND allows. When you are ready to return to VBASICA, enter the DOS command EXIT.

EXAMPLE:

```
SHELL (get a new COMMAND)
A>DIR (you type DIR to see files)
A>EXIT (you type EXIT to return to VBASICA)
Ok (now you are back in VBASICA)
```

The following example writes some data to sort, uses SHELL to sort it, then reads the sorted data to write a report:

```
900 OPEN "SORTIN.DAT" FOR OUTPUT AS 1
950 REM ** write data to be sorted
1000 CLOSE 1
1010 SHELL "SORT SORTIN.DAT SORTOUT.DAT"
1020 OPEN "SORTOUT.DAT" FOR INPUT AS 1
1030 REM ** Process the sorted data

10 SHELL "DIR | SORT > FILES."
20 OPEN "FILES." FOR INPUT AS 1
```

SIN Function

FORMAT:

SIN(X)

PURPOSE:

Returns the sine of X, where X is in radians.

REMARKS:

$\text{COS}(X) = \text{SIN}(X + 3.14159/2).$

EXAMPLE:

```
PRINT SIN(1.5)
```

yields

.9974951

See also the COS function.

SOUND Statement

FORMAT:

SOUND < frequency > , < duration >

PURPOSE:

Generates a sound from the speaker of a specified frequency for a specified duration.

REMARKS:

< frequency > is the desired frequency in Hertz. It is a valid numeric expression returning an unsigned integer from 37 to 32767.

< duration > is the desired duration in clock ticks. It is a valid numeric expression returning an unsigned integer from 0 to 65535. Currently, clock ticks occur 18.2 times per second. If the duration is zero, any currently playing sound is turned off; if no sound is playing there is no effect.

Sounds can be buffered to prevent execution from stopping when VBASICA encounters a new SOUND statement. See the MB command in PLAY.

EXAMPLE:

```
2500 SOUND RND*1000+37,2    'creates random sounds.
```

SPACE\$ Function

FORMAT:

SPACE\$ (I)

PURPOSE:

Returns a string of spaces of length I.

3

REMARKS:

The expression I is rounded to an integer and must be from 0 to 255.

EXAMPLE:

```
10 FOR I=1 TO 5
20 X$=SPACE$(I)
30 PRINT X$;I
40 NEXT I
```

yields

```
1
 2
 3
 4
 5
```

Also see the SPC function.

SPC Function

FORMAT:

SPC(n)

PURPOSE:

Skips spaces in a PRINT statement. n is the number of spaces skipped.

REMARKS:

You can only use SPC with PRINT and LPRINT statements. n must be from 0 to 255. A semicolon (;) is assumed to follow the SPC(n) command.

EXAMPLE:

```
PRINT "OVER" SPC(15) "THERE"
```

yields

```
OVER                THERE
```

Also see the SPACE\$ function.

SQR Function

FORMAT:

SQR(X)

PURPOSE:

Returns the square root of X.

3

REMARKS:

X must be greater than or equal to 0.

EXAMPLE:

```
10 FOR X = 10 to 25 STEP 5
20 PRINT X, SQR(X)
30 NEXT X
```

yields

```
10  3.162278
15  3.872984
20  4.472136
25  5
```

STOP Statement

FORMAT:

STOP

PURPOSE:

Terminates program execution and returns to command level.

REMARKS:

3

You can use STOP statements anywhere in a program to terminate execution. STOP is often used for debugging. When VBASICA encounters a STOP, the following message is printed:

Break in line nnnnn

The STOP statement doesn't close files.

VBASICA always returns to command level after a STOP is executed. Resume execution by issuing a CONT command.

EXAMPLE:

```
10 INPUT A,B,C
20 K = A^2*5.3:L = B^3/.26
30 STOP
40 M = C*K + 100:PRINT M
```

yields

```
? 1,2,3      (you enter 1,2,3)
BREAK IN 30
Ok
```

```
PRINT L      (you enter PRINT L)
30.76923
Ok
CONT        (you enter CONT)
115.9
```

STR\$ Function

FORMAT:

STR\$(n)

PURPOSE:

Returns a string representation of the value of n.

3

EXAMPLE:

```
5 REM ARITHMETIC FOR KIDS
10 INPUT "TYPE A NUMBER";N
20 ON LEN(STR$(N)) GOSUB
30 100,200,300,400,500
```

Also see the VAL function.

STRING\$ Function

FORMAT:

STRING\$(I,J)

STRING\$(I,X\$)

PURPOSE:

Returns a string of length I whose characters all have ASCII code J or the first character of X\$.

3

EXAMPLE:

This example:

```
10 DASH$ = STRING$(10,45)
20 PRINT DASH$;"MONTHLY REPORT";DASH$
```

yields

-----MONTHLY REPORT-----

This example:

```
10 LET A$ = "HOUSTON"
20 LET X$ = STRING$(8,A$)
30 PRINT X$
```

yields

HHHHHHHH

SWAP Statement

FORMAT:

SWAP < var1 > , < var2 >

PURPOSE:

Exchanges the values of two variables.

REMARKS:

Any type variable (integer, single precision, double precision, string) can be SWAPped. Both variables must be of the same type; otherwise, a "Type mismatch" error results.

EXAMPLE:

```
LIST
10 A$= "ONE" :B$= "ALL" : C$= "FOR"
20 PRINT A$ C$ B$
30 SWAP A$, B$
40 PRINT A$ C$ B$
RUN
Ok
ONE FOR ALL
ALL FOR ONE
Ok
```